Fast Time and Space Parallel Algorithms for Solution of Parabolic Partial Differential Equations

Amir Fijany

Jet propulsion Laboratory, California Institute of Technology
Pasadena, CA 91109

Abstract- In this paper, fast time- and space-parallel algorithms for solution of linear parabolic PDEs are developed. It is shown that the seemingly strictly serial iterations of the time-stepping procedure for solution of the problem can be completely decoupled. This decoupling is achieved by using a transformation based on the eigenvalue-eigenvector decompositions of the matrices involved in the iterations and results in time-parallel algorithms that enable the solution for all the time steps to be computed in parallel. The time-parallel algorithms also allow a massively parallel solution of the problem through exploitation of parallelism in space. With a sufficient number of processors, parallelism in both time and space can be fully exploited, leading to a computational complexity of $\max(O(\text{Log } N), O(\text{Log } M)) + O(\text{Log } N)$ for a both time-and space-parallel solution of the problem where M and N stand for the number of time steps and the size of grid. However, for many practical cases, the complexity of time-parallel algorithms is independent of M. The time-parallel algorithms have a highly decoupled structure and hence can be efficiently implemented on the emerging massively parallel MIMD architectures with a minimum communication and synchronization overhead.

Index Terms- Crank-Nicholson method, MIMD parallel architectures, parabolic PDEs, parallel algorithms, time- and space-parallel computation.

1. Introduction

The solution of parabolic PDEs arises in many scientific applications. Therefore, the development of fast and accurate algorithms for the problem has been extensively studied in the literature. The advent of massively parallel architectures offers a new opportunity for a faster solution of the problem. However, in order to fully exploit the computing power of these new architectures, the existing algorithms must be reexamined based on their efficiency for parallel implementation and eventually new algorithms must be developed that, from the onset, take a greater advantage of the massive parallelism.

In this paper, we consider the linear parabolic equation on a bounded domain Ω (which can be one-, two-, or three-dimensional) with boundary Ω' as

$$\frac{\partial U}{\partial t} = \alpha^2 \nabla^2 U \qquad \text{in } \Omega \text{ and } T>t>0 \tag{1}$$

with boundary and initial conditions as

$$U = g \qquad \text{on } \Omega' \text{ and } T>t>0$$

$$U = f \qquad \text{in } \Omega \text{ and } t = 0$$

where ∇^2 is the Laplace operator and α is constant. We first consider the problem with Dirichlet boundary condition. The extension of the results to Neumann boundary condition is discussed later. Also, for two- and three-dimensional cases, Ω is assumed to be regular, i.e., a square or a cube.

The discretization of Eq. (1) by superimposing a uniform grid on Ω and using the usual finite-different schemes leads to a family of iterative methods given by

$$(1 + 2\gamma\delta M_L)U^{(i)} = (I - 2\delta(1 - \gamma)M_L)U^{(i-1)} \qquad i = 1, \ldots, M \tag{2}$$

where I is the unit matrix of appropriate size, M_L is the matrix arising from the discretization of Laplace operator, δ is constant, $\Delta t = \tau$ is the time step size, and $M = T/\tau$.

The three methods for the problem are characterized by the parameter γ as

$\gamma = 0$: Explicit method and Eq. (2) becomes

$$u^{(1)} = (I - 2\delta M_L)U^{(i-1)} \qquad i = 1, \ldots, M \tag{3}$$

γ = 1: Implicit method and Eq. (2) becomes

$$(I + 2\delta M_L)U^{(i)} = U^{(i-1)} \qquad i = 1, \ldots, M \tag{4}$$

γ = 1/2: Crank-Nicholson (C-N) method and Eq. (2) becomes

$$(I + \delta M_L)U^{(i)} = (I - \delta M_L)U^{(i-1)} \qquad i = 1, \ldots, M \tag{5}$$

The explicit method is conditionally stable while both the implicit and C-N methods are unconditionally stable. The C-N method is usually preferred since it is second-order accurate in time while the implicit method is only first-order accurate. This better accuracy in time is achieved at a cost of an additional matrix-vector multiplication. It seems, however, less noticed that the matrix-vector multiplication can be avoided by rewriting Eq. (5) as

$$(I + \delta M_L)((J^{(i)} + U^{(i-1)}) = 2U^{(i-1)} \qquad i = 1, \ldots, M$$

which, by defining $W^{(i)} = U^{(i)} + U^{(i-1)}$, allows to express the C-N method in a modified form as

$$(I/2 + (\delta/2)M_L)W^{(i)} = U^{(i-1)} \tag{6}$$

$$U^{(i)} = W^{(i)} - U^{(i-1)} \tag{7}$$

For both serial and space-parallel (see below) computation, Eqs. (6)-(7) are more efficient than Eq. (5) since the matrix-vector multiplication is replaced by a simple vector addition. However, for our approach Eq. (5) is more suitable since not only it does not increase the computation cost but also it can be better parallelized.

The iterations in Eqs. (3)-(5) represent the time-stepping, or marching in time, procedure for solution of the problem, From a computational point of view, the problem is both time (i.e., number of time steps, M) and space (i.e, size of grid, N) dependent. Throughout this paper, the term space-parallel is used for the, algorithms that exploit parallelism only in each solution of Eqs. (3)-(5) while the term time-parallel refers to those algorithms that exploit full or partial parallelism in computation of all vectors $U^{(i)}$, $i = 1, 2, \ldots, M$. A both time- and space-parallel algorithm is then the one that exploits full or partial parallelism in computation of all vectors $U^{(i)}$

as well as parallelism in computation of each vector.

The coefficient matrices in Eqs. (4)-(6) have a symmetric, positive-definite, and sparse structure. This allows the use of the rather generic iterative methods-such as SOR, conjugate gradient, etc. [11, for solution of the linear systems in Eqs. (4)-(6). These matrices have additional structures similar to those arising in solution of Poisson Equation. In this sense, Eqs. (4)-(6) represent a sequence of Poisson Equations. Therefore, the so called fast Poisson Solvers can be used for the direct solution of the linear systems in Eqs. (4)-(6) with a greater computational efficiency over the iterative methods [2]. The application of serial and parallel fast Poisson Solvers are discussed in more detail in Sections II-IV.

The fact that the application of the implicit and C-N methods requires the solution of a linear system at each time step has been considered as a limiting factor for efficient parallel computation [3]. To overcome this limitation and increase the efficiency for parallel computation, two different approaches have been proposed. Gallopoulos and Saad [4] and Serbin [5] have developed implicit methods wherein partial fraction decomposition technique is employed to better parallelize the linear system solution. The explicit method, while limited in its range of stability, is highly efficient for parallel and vector computation since it only involves a sequence of matrix-vector multiplications. Motivated by this greater efficiency for parallel and vector computation, Rodrigue [6], Rodrigue and Wolitzer [7], and Evans [8] have developed new explicit methods with greater stability regions. However, these approaches result in algorithms that can be classified as space-parallel since they attempt to parallelize the computation of each iteration while the overall computation in time remains strictly serial.

In fact, it seems that the time-stepping procedure in Eqs. (3)-(5) implies a strict sequentiality of the computation in time. This has motivated the development of new approaches to increase parallelism in time. Waveform

relaxation [9] and windowed relaxation [10] methods have been developed to increase time-parallelism in the computation while using the iterative techniques such as Jacobi, Gaus-Seidel, and SOR for the linear system solution, Womble [11] has proposed the parallel time-stepping method in which, while the exact solution for one time step is' computed by a group of processors, other processors can compute a good initial guess for the next time steps based on partial solutions of previous time steps. Time-parallel algorithms based on the multigrid method have been proposed by Hacbusch [12] and Horton and Knirsch [131 wherein parallelism in time is achieved by performing the computation for several time steps simultaneously. However, these time-parallel algorithms achieve a rather limited parallelism in time. In fact, Womble [111 supports the assessment of [141 wherein simultaneous solution for all the time steps is not considered practical.

In this paper we develop time-parallel algorithms that, for the class of problems defined by Eq. (l), allow the iterations in Eqs. (3)-(5) to be completely decoupled and performed in parallel [15]. The main emphasis, however, is on the time-parallel computation of the C-N method. The decoupling is achieved by transforming Eqs. (3)-(5) into a diagonal form. This transformation, that is based on the eigenvalue-eigenvector decomposition of the matrices involved in the equations, reduces Eqs. (3)-(5) to a set of First Order Homogeneous Linear Recurrences (FOHLR) which then allow the solution for all the time-steps to be computed in parallel. Our results clearly prove that, unlike the general assumption, the iterations in Eq. (3)-(5) can be more efficiently parallelized in time than in space. As a result, even with a limited number of processors, it is more efficient to exploit parallelism in time than in space.

It should be pointed out that, for most cases considered in this paper, the eigenvalue-eigenvector decomposition of these matrices have been well

known, However, such a knowledge has been usually used for analyzing the stability of the different methods (see for example [16]) rather than deriving algorithms for solution of problem. This can be explained by the fact that, at first glance, it seems that the use of eigenvalue-eigenvector decomposition results in inefficient algorithms for serial computation. However, while this is true for the one- and two-dimensional cases, the resulting algorithms not only are highly efficient for parallel solution of problem with any dimension but also, for the three-dimensional case, seem to be the most efficient even for serial computation,

This paper is organized as follows. In Sections II-IV the time-parallel algorithms for one-, two-, and three-dimensional problems are developed. The extension to Neumann boundary condition and higher-order finite-difference schemes are presented in Section V. The practical implementation of the time-parallel algorithms is discussed in Section VI. Finally some concluding remarks are made in Section VII.

II. One-Dimensional Parabolic Equation

A. Problem Statement and Crank-Nicholson Method

For one-dimensional case, we consider Ω to be of unit length (i.e. , a rod of length 1). The parabolic equation is given as

$$\frac{\partial U}{\partial t} = \alpha^2 \frac{\partial^2 U}{\partial x^2} \qquad x\varepsilon\Omega \text{ and } T>t>0$$

with boundary and initial conditions as

$$U(t,x) = g(x) \qquad x\varepsilon\Omega' \text{ and } T>t>0$$

$$U(0,x) = f(x) \qquad x\varepsilon\Omega$$

Superimposing a uniform grid of size $\Delta x = h$ and using the 3-point central difference scheme, the C-N method is then given by

$$(I_N + \delta M_{L1})U^{(i)} = (I_N - \delta M_{L1})U^{(i-1)} \qquad i = 1, \ldots, M \tag{8}$$

where $h = 1/(N+1)$, $\delta = \alpha^2\tau/2h^2$, I_N is the NxN unit matrix, and $M_{L1} \varepsilon R^{NxN}$ is a

tridiagonal matrix as M_{L1} = Tridiag[-1, 2, -11.

As stated before, for both serial and space-parallel computation, the modified C-N method given by Eqs. (6)-(7) is more efficient than the direct solution of Eq. (8). Further computational efficiency in both serial and parallel solution of Eq. (6) can be achieved by exploiting the special Toeplitz structure of the coefficient matrix (see for example [17,181).

In any case, the cost of serial computation of Eqs. (6)-(7) is of O(N). This leads to a total serial computational complexity of $O(MN)$ for the problem, With O(N) processors, the solution of the tridiagonal system in Eq. (6) can be obtained in $O(\text{Log } N)$ steps by using, for example, the parallel algorithms in [18,191 while the computation of Eq. (7) can be performed in O(1) step. This leads to a parallel computational complexity of $O(M\text{Log } N)$ which indicates that

a) The computation is fully parallelized in space, i.e. , the computation of each step is fully parallelized, wherein the time lower bound of $O(\text{Log } N)$ is achieved, and

b The computation is strictly serial in time.

B Time-Parallel Algorithm

Following theorem is used in the derivation of the time-parallel algorithm.

Theorem 1. The eigenvalue-eigenvector decomposition of a symmetric tridiagonal Toeplitz matrix T = Tridiag[b, a, b]εR^{NxN} is given as

(9) $$T = Q\lambda Q$$

where the matrix $Q \triangleq \text{Row}\{Q^{(i)}\}\varepsilon R^{NxN}$ is the set of normalized eigenvectors of matrix T with $Q^{(i)} = (2/N+1)^{1/2}\text{Col}\{\sin(ij\pi/N+1)\}\varepsilon R^{N}$, i and j = 1, 2, ... N, being the ith eigenvector. The diagonal matrix $\lambda = \text{Diag}\{\lambda_i\}\varepsilon R^{NxN}$ is the set of eigenvalues of matrix T with $A_i = a + 2b\cos(i\pi/N+1)$ being the ith eigenvalue.

Proof. See for example [20, p.349]. 0

Note that, Q is a symmetric orthonormal matrix, i.e., $Q = Q^t = Q^{-1}$. From the

above theorem, the eigenvalue-eigenvector decomposition of M_{L1} is given as

(10) $$A_{L1} = Q\lambda_1 Q$$

where $\lambda_1 \triangleq \text{Diag}\{\lambda_{1i}\} \in R^{NxN}$, i = 1, ... N, with

(11) $$\lambda_{1i} = 2 - 2\cos(i\pi/N+1)$$

Replacing Eq. (10) into Eq. (8), we get

(12) $$Q(I_N + \delta\lambda_1)QU^{(i)} = Q(I_N - \delta\lambda_1)QU^{(i-1)} \qquad i = 1, 2, \ldots, M$$

Let $\tilde{U}^{(i)} \triangleq QU^{(i)}$ and $D_1 \triangleq (I_N + \delta\lambda_1)^{-1}(I_N - \delta\lambda_1)$. Since Q is an orthonormal and hence a nonsingular matrix, it then follows that

(13) $$\tilde{U}^{(i)} = D_1\tilde{U}^{(i-1)} \qquad i =1, 2, \ldots, M$$

which represents a FOHLR. However, Eq. (13) implies that

(14) $$\tilde{U}^{(i)} = (D_1)^i\tilde{U}^{(0)} \qquad i =1, 2, \ldots, M$$

The diagonal matrix D_1 is a function of problem size and time and space discretization parameters. If a same problem is solved many times for different boundary and/or initial conditions, then all the matrices $(D_1)^i$ can be precomputed (see also Sec. VI), In this case, starting with $\tilde{U}^{(0)}$ all $\tilde{U}^i$ can be computed in parallel from Eq. (14). Assuming that all matrices $(D_1)^i$ are precomputed, the time-parallel algorithm is then given as

Step I: $$\tilde{u}^{(0)} = QU^{(0)}$$

Step II: $$\tilde{\tilde{u}}^{(i)} = (D_1)^i\tilde{U}^{(0)} \qquad i = 1, 2, \ldots, M$$

Step 111: $$u^{(i)} = Q\tilde{U}^{(i)} \qquad i = 1, 2, \ldots, M$$

C. Comparison of Serial, Space-Parallel, and Time-Parallel Algorithms

The computational structure of the time-parallel algorithm is shown in Fig. la. As can be seen, the computations in Steps II-III are completely decoupled and can be performed in parallel for all i = 1, 2, ... M, In this case, the computational complexity of time-parallel algorithm, unlike that of serial and space-parallel algorithms, is independent of M. That is, the O(M) dependency is reduced to O(l), However, as discussed below, care should be taken in analyzing the performance of the time-parallel algorithm,

The matrix Q is the one-dimensional Discrete Sine Transform (DST) operator. Therefore, the multiplication of any vector by the matrix Q in Steps I and 111 is tantamount to performing a one-dimensional DST which, by using the fast techniques, can be computed in $O(N \text{Log } N)$ [211. It follows that the complexity of serial implementation of the time-parallel algorithm is of $O(MN \text{Log } N)$. This implies that the time-parallel algorithm is not efficient for serial computation. In fact, the algorithm is *asymptotically* inconsistent in the sense defined by Ortega and Voigt [3] since the complexity of its serial implementation, i.e., $O(MN \text{Log } N)$, is greater than that of the best serial algorithm for the problem, i.e., $O(MN)$.

Due to this asymptotic inconsistency, the performance of the time-parallel algorithm strongly depends on the degree to which parallelism is exploited in its computation. Using M processors, that corresponds to a coarse grain parallel implementation and is designated as M-parallel implementation, the computational complexity of the time-parallel algorithm 1s of $O(N \text{Log } N)$. If $M > \text{Log } N$ (which is likely to be the case for many practical applications), then the time-parallel algorithm achieves a speedup of $O(M/\text{Log } N)$ over the best serial algorithm. However, only for $M > N$ the time-parallel algorithm becomes faster than the space-parallel algorithm with a relative speedup of $O(M/N)$. This implies that for problems with small M the time parallel-algorithm may become less efficient than the best space-parallel algorithm.

Using $O(MN)$ processors, that corresponds to a two-level or a both time- and space-parallel implementation, the DSTS in Steps I and III can be performed in $O(\text{Log } N)$ [181 and the computation in Step II in O(1) which leads to an overall computational complexity of $O(\text{Log } N)$. This represents a speedup of $O(MN/\text{Log } N)$ over the best serial algorithm and a relative speedup of O(M) over the best space-parallel algorithm. The latter performance is of particular significance since it indicates that by increasing the number of processors from O(N) to $O(MN)$ a linear relative speedup of O(M) can be achieved.

III. Two-Dimensional Parabolic Equation

A. Statement of Problem and Crank-Nicholson Method

For the two-dimensional case, we consider Ω to be a unit square, The parabolic equation is given as

$$\frac{\partial U}{\partial t} = \alpha^2 \left(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2}\right) \qquad x, y \in \Omega \text{ and } T>t>0$$

with boundary and initial conditions as

$$U(t,x,y) = g(x,y) \qquad x, y \in \Omega' \text{ and } T>t>0$$

$$U(0,x,y) = f(x,y) \qquad x, y \in \Omega$$

Superimposing a uniform grid on Ω ($\Delta x = \Delta y = h$) and using the usual 5-point finite difference scheme, the C-N method is given by

$$(I_{N2} + \delta M_{L2})U^{(i)} = (I_{N2} - \delta M_{L2})U^{(i-1)} \qquad i = 1, 2, \ldots)' \tag{15}$$

where, as before, $h = 1/(N+1)$ and $\delta = \alpha^2\tau/2h^2$. I_{N2} is the N2xN2 unit matrix, $A_{L2} = \text{Tridiag}[-I_N, A, -I_N] \in R^{N^2 x N^2}$ is the block tridiagonal matrix arising from the discretization of two-dimensional Laplace operator, and $A \in R^{NxN}$ is a tridiagonal matrix as $A = \text{Tridiag}[-1, 4, -1]$.

Again, the modified C-N method given by Eqs. (6)-(7) is more efficien than the direct solution of Eq. (15) since the multiplication of a block tridiagonal matrix by a vector is replaced by a vector addition. For this case, the coefficient matrix in Eq. (6) has a similar structure to M_{L2} which is the matrix arising in solution of two-dimensional Poisson Equation. Thus, the fast Poisson Solvers, i.e., Bouneman's Variant of Cyclic Reduction (CR) algorithm [22,23], Matrix Decomposition (MD) algorithm [23,24], and Fourier Analysis (FA) algorithm [251, with a serial complexity of $0(N^2 \text{Log } N)$ can be used for direct solution of Eq. (6). This leads to a complexity of $O(N^2 \text{Log } N)$ for each solution of Eq. (6) and an overall complexity of $0(MN^2 \text{Log } N)$ for the serial computation of problem.

For space-parallel computation, both the MD and FA algorithms are more efficient than the CR algorithm, Sweet [26] and Gallopoulos and Saad [27] have

shown that parallel computation of CR algorithm can be performed in $O(\text{Log}^2 N)$ steps with $O(N^2)$ processors. However, with $O(N^2)$ processors, parallel computation of both MD and FA algorithms can be performed in $O(\text{Log } N)$ steps [18,28]. Therefore, by using the parallel variant of either MD or FA algorithms with $O(N^2)$ processors, each solution of Eq. (6) can be computed in $O(\text{Log } N)$ while the vector addition in Eq. (7) can be performed in 0(1). This leads to an overall computational complexity of $O(M\text{Log } N)$ for space-parallel solution of the problem which indicates that the computation is fully parallelized in space but is strictly serial in time.

B. Time-Parallel Algorithm

The time-parallel algorithm for two-dimensional case is based on the derivation of the eigenvalue-eigenvector decomposition of matrix $\mathcal{M}_{L2}$. To this end, let us first consider a matrix $Q \triangleq \text{Diag}[Q, Q, \ldots, Q, Q] \in \mathbb{R}^{N^2 \times N^2}$. By definition, it follows that Q is a symmetric orthonormal matrix and hence $Q = Q^t = Q^{-1}$. Also, consider a permutation matrix $P \in \mathbb{R}^{N^2 \times N^2}$ as

$$P = \left[\begin{array}{cccccc}
1 & 0 & \cdots & 0 & 0 & \cdots \\
0 & 0 & \cdots & 0 & 0 & \cdots \\
\vdots & & & & & \\
0 & 0 & \cdots & 0 & 0 & \cdots \\
\hline
0 & 1 & \cdots & 0 & 0 & \cdots \\
0 & 0 & \cdots & 0 & 0 & \cdots \\
\vdots & & & & & \\
0 & 0 & \cdots & 0 & 0 & \cdots \\
\hline
\vdots & & & & & \\
\hline
0 & 0 & \cdots & 1 & 0 & \cdots \\
0 & 0 & \cdots & 0 & 0 & \cdots \\
\vdots & & & & & \\
0 & 0 & \cdots & 0 & 0 & \cdots
\end{array}\right]$$

P is a symmetric permutation matrix, i.e., $P = P^t$, and hence $PP = I_{N2}$.

Theorem 2. The matrix $\mathcal{M}_{L2}$ has an eigenvalue-eigenvector decomposition as

$$\mathcal{M}_{L2} = QPQ\lambda_2 QPQ \tag{16}$$

where $\lambda_2 \triangleq \text{Diag}\{\lambda_{21}, \} \in \mathbb{R}^{N^2 \times N^2}$, i and j = 1, 2, ... N, with

(17) $\lambda_{2ij} = 4 - 2\cos(i\pi/N+1) - 2\cos(j\pi/N+1)$

Proof. From Theorem 1, the matrix M_{L2} can be expressed as

(18) $M_{L2} = \text{Tridiag}[-I_N, Q\lambda_1'Q, -I_N] = Q\Lambda_2 Q$

where $\Lambda_2 \in \mathbb{R}^{N^2 \times N^2}$ is a block tridiagonal matrix as $\Lambda_2 = \text{Tridiag}[-I_N, \lambda_1', -I_N]$ with $\lambda_1' = \text{Diag}\{\lambda_{1i}'\} \in \mathbb{R}^{N\times N}$, i =1, 2,. ..,N, and $\lambda_{1i}' = 2 + \lambda_{1i} = 4 - 2\cos(i\pi/N+1)$. The block elements of Λ_2 are diagonal and hence A_2 can be reduce to a block diagonal matrix as

(19) $A_2 = PPA_2PP = P(P\Lambda_2 P)P = PT_2P$

where $T_2 = \text{Diag}\{T_{2i}\} \in \mathbb{R}^{N^2 \times N^2}$ and $T_{2i} = \text{Tridiag}[-1, \lambda_{1i}', -1] \in \mathbb{R}^{N\times N}$, i = 1, 2, ... N. From Eqs. (18)-(19), M_{L2} can be expressed as

(20) $M_{L2} = QPT_2PQ$

The submatrices T_{2i} have a symmetric tridiagonal Toeplitz structure. Therefore, from Theorem 1 and definition of Q and T_2, the eigenvalue-eigenvector decomposition of T_2 is given by

(21) $T_2 = Q\lambda_2 Q$

The eigenvector-eigenvalue decomposition of matrix M_{L2} in Eq. (16), is then obtained by replacing Eq. (21) into Eq. (20). θ

Note that, from the definition of Q and *P*, it fo lows that the matrix $\theta = QPQ$ is symmetric and orthonormal, i.e., $\theta = \theta^t = \theta^{-1}$

The time-parallel algorithm is derived by replacing Eq. (16) into Eq. (15)

(22) $QPQ(I_{N2} + \delta\lambda_2)QPQU^{(i)} = QPQ(I_{N2} - \delta\lambda_2)QPQU^{(i-1)}$

Let $\tilde{U}^{(i)} \triangleq QPQU^{(i)}$ and $D_2 \triangleq (I_{N2} + \delta\lambda_2)^{-1}(I_{N2} - \delta\lambda_2)$. Since the matrix $\theta = QPQ$ is orthonormal and hence nonsingular, it then follows that

(23) $\tilde{U}^{(i)} = D_2\tilde{U}^{(i-1)}$ i =1,2, . . ., M

which implies that

(24) $\tilde{u}^{(i)} = (D_2)^i\tilde{U}^{(0)}$ i = 1, 2, . . ., M

Again, assuming that all the diagonal matrices $(D_2)^i$ are precomputed, the time-parallel algorithm is given as

Step I: $\tilde{U}^{(0)} = QPQU^{(0)} = \theta U^{(0)}$

Step 11. $\tilde{u}^{(1)} = (D_2)^i \tilde{U}^{(0)}$ $i = 1, 2, \ldots, M$

Step III: $u^{(i)} = QPQ\tilde{U}^{(1)} = \theta \tilde{U}^{(1)}$ $i = 1, 2, \ldots, M$

C. Comparison of Serial, Space-Parallel, and Time-Parallel Algorithms

The computational structure of the time-parallel algorithm is shown in Fig. lb. Again, as can be seen, the computations in Steps II and 111 are completely decoupled. If these computations are performed in parallel then the computation complexity of time-parallel algorithm is independent of M.

The matrix $\theta = QPQ$ is the two-dimensional DST operator. Therefore, the multiplication of any vector by matrix θ in Steps I and III is tantamount to performing a two-dimensional DST which, by using the fast techniques, can be performed in $O(N^2 \text{Log } N)$ steps. It then follows that the complexity of serial implementation of the time-parallel algorithm is of $O(MN^2 \text{Log } N)$. This implies that the algorithm is asymptotically consistent, i.e., its serial implementation is, asymptotically, as fast as the best serial algorithm for the problem. In terms of actual number of operations, the time-parallel algorithm is also competitive with the best serial algorithms (see Sec. VI).

For the two-dimensional problem, due to this asymptotic consistency, the time-parallel algorithm is always efficient regardless of the parallel implementation strategy and the number of processors employed. With M processors, the computational cost of time-parallel is of $O(N^2 \text{Log } N)$ which represents a linear speedup of **O(M)** over the best serial algorithm. By using $O(MN^2)$ processors, i.e., a two-level parallel implementation, parallelism in both time and space can be fully exploited. In this case, the computations in Steps I and 111 can be performed in $O(\text{Log } N)$- since a two-dimensional DST consists of two steps wherein at each step N decoupled one-dimensional DST'S are computed- and the computation in Step 11 can be computed in O(l). This leads to a computational cost of $O(\text{Log } N)$ for both time- and space-parallel

algorithm. Compared with the best space-parallel algorithm, this represents a relative speedup of O(M) at a cost of an increase of O(M) in the number of processors which indicates a very high processors utilization factor,

IV. Three-Dimensional Parabolic Equation

A. Statement of Problem and Crank-Nicholson Method

For the three-dimensional case, we consider the parabolic equation in a unit cube domain as

$$\frac{\partial U}{\partial t} = \alpha^2 \left(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2}\right) \qquad x, y, z \in \Omega \text{ and } T>t>0$$

with boundary and initial conditions as

$$U(t,x,y,z) = g(x,y,z) \qquad x, y, z \in \Omega' \text{ and } T>t>0$$

$$U(0,x,y,z) = f(x,y,z) \qquad x, y, z \in \Omega$$

Superimposing a uniform grid on Ω ($\Delta x = \Delta y = \Delta z = h$) and using the usual 7-point finite difference approximation, the C-N method is given by

$$(I_{N3} + \delta M_{L3})U^{(i)} = (I_{N3} - \delta M_{L3})U^{(i-1)} \tag{25}$$

where, as before, $h = 1/(N+1)$ and $\delta = \alpha^2\tau/2h^2$. I_{N3} is the N3xN3 unit matrix, $M_{L3} = \text{Tridiag}[-I_{N2}, B, -I_{N2}] \in R^{N^3 \times N^3}$ is the block tridiagonal matrix arising from the discretization of three-dimensional Laplace operator, and $B = \text{Tridiag}[-I_N, A', I_N] \in R^{N^2 \times N^2}$ is a block tridiagonal matrix with $A' = \text{Tridiag}[-1, 6, -1] \in R^{N \times N}$.

Again, the modified C-N method given by Eq. (6)-(7) is more efficient than the direct solution of Eq. (25). For this case, the coefficient matrix in Eq. (6) has a structure similar to M_{L3} which is the matrix arising in solution of the three-dimensional Poisson Equation (see for example [29]). Therefore, the fast Poisson Solvers for the three-dimensional problem can be used for direct solution of Eq. (6).

The extension of the CR and MD algorithms to the solution of three-dimensional Poisson Equation has been reported in [29,30,31]. The analysis in [30] indicates that the CR and MD algorithms have a serial computational

complexity of 0(N^3Log^2N) and O(N^3Log N), respectively. There seems to be no report on the extension of the FA algorithm. However, it is rather straightforward to show that the algorithm can be extended to the solution of three-dimensional problem with a computational complexity of O(N^3Log N). Note that, for the three-dimensional problem, both the MD and FA algorithms are faster than the CR algorithm.

There are very few reports on parallel computation of fast Poisson Solvers for the three-dimensional case. Sameh [291 and Sweet *etal* [31] have studied parallel computation of the MD algorithm. Both works are concerned with some specific parallel implementation of the algorithm wherein limited parallelism is exploited. However, based on the analyses in [29,311, it is straightforward to show that, by using $O(N^3)$ processors, the time lower bound of O(Log N) can be achieved in parallel computation of the MD algorithm. It can also be shown that the same bounds on time and processors are achievable in parallel computation of the FA algorithm, However, the analysis of parallelism in computation of the CR algorithm seems to be less straightforward. In fact, a first step, the extension of the works in [26,271 to the three-dimensionai case needs to be studied, Therefore, only parallel variants of the MD and FA algorithms are considered here, We can conclude that, by using $O(N^3)$ processors and parallel variant of either the MD or FA algorithms, each solution of Eq. (6) can be computed in O(Log N) while the vector addition in Eq. (7) can be done in O(l). This leads to a complexity of O(MLog N) for space-parallel solution of the problem which, again, indicates that the computation is fully parallelized in space but is strictly serial in time.

B. Time-Parallel Algorithm

The time-parallel algorithm for three-dimensional case is also based on the derivation of the eigenvalue-eigenvector decomposition of matrix M_{L3}. To this end, let us first consider two matrices Q and Θ as

$Q \triangleq \text{Diag}[Q, Q, \ldots, Q, Q] \in \mathbb{R}^{N^3 \times N^3}$ and $\Theta \triangleq \text{Diag}[\theta, \theta \ldots \theta, \theta] \in \mathbb{R}^{N^3 \times N^3}$

From their definition, it follows that Q and θ are symmetric orthonormal matrices, i.e., $Q = Q^t = Q^{-1}$ and $\Theta = \Theta^t = \Theta^{-1}$. Also, consider a permutation matrix $\mathcal{P} \in \mathbb{R}^{N^3 \times N^3}$ that has a structure similar to P but each of its block has N rows and N^2 columns. Note that, unlike P, the matrix $\mathcal{P}$ is not symmetric. But $\mathcal{P}\mathcal{P}^t = \mathcal{P}^t\mathcal{P} = I_{N3}$ since $\mathcal{P}$ is a permutation matrix.

Theorem 3. The matrix M_{L3} has an eigenvector-eigenvalue decomposition as

$$M_{L3} = \Theta\mathcal{P}^t Q \lambda_3 Q \mathcal{P} \Theta \tag{26}$$

where $\lambda_3 \triangleq \text{Diag}\{\lambda_{3ijk}\} \in \mathbb{R}^{N^3 \times N^3}$, i, j, and k = 1, . . ., N, with

$$\lambda_{3ijk} = 6 - 2\cos(i\pi/N+1) - 2\cos(j\pi/N+1) - 2\cos(k\pi/N+1) \tag{27}$$

Proof. From Theorem 2, the matrix M_{L3} can be expressed as

$$M_{L3} = \text{Tridiag}[-I_{N2}, \Theta\lambda_2'\Theta, -I_{N2}] = \Theta\Lambda_3\Theta \tag{28}$$

where $\lambda_2' = 21_{N2} + \lambda_2$ and $\Lambda_3 \in \mathbb{R}^{N^3 \times N^3}$ is a block tridiagonal matrix as $\Lambda_3 = \text{Tridiag}[-I_{N2}, \lambda_2', -I_{N2}]$. The block elements of A_3 are diagonal and hence A_3 can be reduced to a block diagonal matrix as

$$A_3 = \mathcal{P}^t\mathcal{P}\Lambda_3\mathcal{P}^t\mathcal{P} = \mathcal{P}^t(\mathcal{P}\Lambda_3\mathcal{P}^t)\mathcal{P} = \mathcal{P}^t T_3 \mathcal{P} \tag{29}$$

where $T_3 \triangleq \text{Diag}\{T_{3ij}\} \in \mathbb{R}^{N^3 \times N^3}$, i and j = 1, 2, ,.., N, and $T_{3ij} = \text{Tridiag}[-1, A'_{2ij}, -1] \in \mathbb{R}^{N \times N}$. From Eqs. (28)-(29), M_{L3} is expressed as

$$M_{L3} = \Theta\mathcal{P}^t T_3 \mathcal{P}\Theta \tag{30}$$

The tridiagonal matrices T_{31j} are symmetric and Toeplitz. Therefore, from Theorem 1 and definition of Q and T_3, the eigenvalue-eigenvector decomposition of T_3 is given by

$$T_3 = Q\lambda_3 Q \tag{31}$$

The eigenvector-eigenvalue decomposition of matrix M_{L3} in Eq. (26), is then obtained by replacing Eq, (31) into Eq. (30). □

Note that, the matrix $\Phi = Q\mathcal{P}\Theta$ is not symmetric but it is orthogonal since

$$\Phi\Phi^t = (Q\mathcal{P}\Theta)(\Theta\mathcal{P}^t Q) = I_{N3}$$

The time-parallel algorithm is derived by replacing Eq. (26) into Eq. (25)

(32) $$\Theta\mathcal{P}^t Q(I_{N3} + \delta\lambda_3)Q\mathcal{P}\Theta U^{(i)} = \Theta\mathcal{P}^t Q(I_{N3} - \delta\lambda_3)Q\mathcal{P}\Theta U^{(i-1)}$$

Let $\tilde{U}^{(i)} \triangleq Q\mathcal{P}\Theta U^{(i)}$ and $D_3 \triangleq (I_{N3} + \delta\lambda_3)^{-1}(I_{N3} - \delta\lambda_3)$. Multiplying both sides of Eq. (32) by the nonsingular matrix $\Phi = Q\mathcal{P}\Theta$ gives

(33) $$\tilde{u}^{(i)} = D_3\tilde{U}^{(i-1)}$$

which implies that

(34) $$u'^{(i)} = (D_3)^i\tilde{U}^{(0)}$$

Again, assuming that all the diagonal matrices $(D_3)^i$ are precomputed, the time-parallel algorithm is given as

Step 1: $\tilde{u}^{(0)} = Q?'GU^{(o)} = \Phi U^{(0)}$

Step II: $\tilde{U}^{(i)} = (D_3)^i\tilde{U}^{(0)}$ $\quad i = 1, 2, \ldots, M$

Step III: $U^{(i)} = \Theta\mathcal{P}^t Q\tilde{U}^{(i)} = \Phi^t\tilde{U}^{(i)}$ $\quad i = 1, 2, \ldots, M$

C. Comparison of Serial, Space-Parallel, and Time-Parallel Algorithms

The computational structure of the time-parallel algorithm is shown in Fig. 1c. Again, the computations in Steps II and 111 are completely decoupled. If these computations are performed in parallel then the computational complexity of time-parallel algorithm is independent of M.

The matrices $\Phi = \Theta\mathcal{P}Q$ and $\Phi^t = \Theta\mathcal{P}^tQ$ are the operators for tree-dimensional direct and inverse DST. Therefore, each matrix-vector multiplication in Steps I and III is equivalent to performing a three-dimensional DST which, by using the fast techniques, can be computed in $0(N^3 \text{Log } N)$ steps. It follows that the complexity of serial implementation of the time-parallel algorithm is of $O(MN^3 \text{Log } N)$ which indicates the asymptotic consistency of the algorithm. Interestingly, in terms of total number of operations, the time-parallel algorithm is even more efficient than the MD and FA algorithms for serial solution of the problem (see Sec. VI)

Due to the asymptotic consistency and also the efficiency in terms of the total number of operations, the time-parallel algorithm for three-dimensional case is highly efficient regardless of the parallel implementation strategy

and the number of processors employed. With M processors, the computational cost of time-parallel algorithm is of $O(N^3 \text{Log } N)$ which indicates a linear speedup of O(M) over the best serial algorithm. By using $O(MN^3)$ processors, i.e., a two-level. parallel implementation, parallelism in both time and space can be fully exploited. In this case, the computations in Steps I and III can be performed in $O(\text{Log } N)$- since a three-dimensional DST consists of two steps wherein at each step N^2 decoupled one-dimensional DSTS need to be computed- and the computation in Step II can be computed in O(l). This leads to a computational cost of $O(\text{Log } N)$ for both time- and space-parallel algorithm. Compared with the best space-parallel algorithm, this represents a relative speedup of O(M) at a cost of an increase of O(M) in the number of processors which indicates a very high processors utilization factor.

V. Some Extensions of Time-Parallel Algorithm

A. Explicit and Implicit Methods

Our approach can be also applied to parallelize the iterations of the explicit and implicit methods by simply replacing the eigenvalue-eigenvector decomposition of appropriate matrix M_L in Eqs. (2) and (3), The resulting time-parallel algorithms will have exactly the same computational structure and cost as those for the C-N method with the only exception that the diagonal matrices D_j, $j = 1, 2, 3$, will be different.

B. Neumann Boundary Condition

A frequently arising type of boundary condition in solution of parabolic PDEs is the Neumann type wherein the normal derivative, $\partial U/\partial n$, is specified on the boundary, In order to develop time-parallel algorithms for solution of the problem with Neumann boundary condition, let us consider the two-dimensional problem of Sec. 111. With the Neumann boundary condition, the discretization of two-dimensional Laplace operator results in a block tridiagonal matrix $\hat{M}_{L2} \in \mathbb{R}^{N^2 \times N^2}$ as [231

$$\hat{M}_{L2} = \begin{bmatrix} \hat{A} & -2I_N & & \\ -I_N & \hat{A} & -I_N & \\ & -I_N & \hat{A} & -I_N \\ & & -2I_N & \hat{A} \end{bmatrix}$$

with the submatrix $\hat{A} \varepsilon \mathbb{R}^{N \times N}$ given as

$$\hat{A} = \begin{bmatrix} 4 & -2 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -2 & 4 \end{bmatrix}$$

The time-parallel algorithm is derived by developing the eigenvalue-eigenvector decomposition of matrices $\hat{A}$ and $\hat{M}_{L2}$ as follows.

Theorem 4. The eigenvalue-eigenvector decomposition of a tridiagonal matrix

$$\hat{T} = \begin{bmatrix} a & 2b & & \\ b & a & b & \\ & \ddots & \ddots & \ddots \\ & & b & a & b \\ & & & 2b & a \end{bmatrix}$$

is given as

$$\hat{T} = \hat{Q}\hat{\lambda}\hat{Q}^{-1} \tag{35}$$

where the matrix $\hat{Q} \triangleq \text{Row}\{\hat{Q}^{(i)}\} \varepsilon \mathbb{R}^{N \times N}$, $i = 0, 1, \ldots, N-1$, is the set of eigenvectors of matrix $\hat{T}$ with $\hat{Q}^{(i)} = \text{Col}\{\cos(ij\pi/N-1)\} \varepsilon \mathbb{R}^{N}$, $j = 0, 1, \ldots, N-1$, being the ith eigenvector. The diagonal matrix $\hat{\lambda} \triangleq \text{Diag}\{\lambda_i\} \varepsilon \mathbb{R}^{N \times N}$ is the set of eigenvalues of matrix $\hat{T}$ with $\hat{\lambda}_i = a + 2b\cos(i\pi/N-1)$ being the ith eigenvalue.

Proof. First, consider a specific matrix $\hat{T}'$ for which a = 2 and b = -1. For matrix $\hat{T}'$, Van Loan [21, p.252] has shown that the eigenvectors are given by matrix $\hat{Q}$ and the eigenvalues by a diagonal matrix $\hat{\lambda}' = \text{Diag}\{\hat{\lambda}'_i\} \varepsilon \mathbb{R}^{N \times N}$ where

$$\hat{\lambda}'_i = 2 - 2\cos(i\pi/N-1) \qquad i = 0, 1, \ldots, N-1$$

The result of [211 can be easily generalized by noting that any matrix $\hat{T}$ can be written in terms of $\hat{T}'$ as

(36) $$\hat{T} = (a + 2b)I_N - b\hat{T}' = \hat{Q}((a + 2b)I_N - b\hat{\lambda}')\hat{Q}^{-1}$$

which implies that

$$\hat{\lambda} = (a + 2b)I_N - b\hat{\lambda}' \Rightarrow \hat{\lambda}_i = (a + 2b) - b\hat{\lambda}_i' = a + 2b\cos(i\pi/N-1) \quad \square$$

The eigenvalues of submatrix $\hat{A}$, for which $a = 4$ and $b = -1$, are then given as

(37) $$\hat{\lambda}_1 = \text{Diag}\{\hat{\lambda}_{1i}\} \text{ with } \hat{\lambda}_{1i} = 4 - 2\cos(i\pi/N-1) \qquad i = 0, 1, \ldots N-1$$

The eigenvalue-eigenvector decomposition of matrix $\hat{M}_{L2}$ is derived in a similar way as that of M_{L2} and by using the result of Theorem 4.

Theorem 5. The eigenvalue-eigenvector decomposition of matrix $\hat{M}_{L2}$ is given as

(38) $$\hat{M}_{L2} = \hat{Q}P\hat{Q}\hat{\lambda}_2\hat{Q}^{-1}P\hat{Q}^{-1}$$

where $\hat{Q} \triangleq \text{Diag}[\hat{Q}, \hat{Q}, \ldots, \hat{Q}, \hat{Q}] \in \mathbb{R}^{N^2 \times N^2}$ and $\hat{\lambda}_2 \triangleq \text{Diag}\{\hat{\lambda}_{2ij}\} \in \mathbb{R}^{N^2 \times N^2}$, for i and j = 0, 1, ..., N-1, with

(39) $$\hat{\lambda}_{2ij} = 4 - 2\cos(i\pi/N-1) - 2\cos(j\pi/N-1)$$

Proof. From Theorem 4 and Eq. (37), $\hat{M}_{L2}$ is written as

(40) $$\hat{M}_{L2} = \hat{Q}\hat{\Lambda}_2\hat{Q}^{-1}$$

where $\hat{\Lambda}_2$ is a block tridiagonal matrix given as

$$\hat{\Lambda}_2 = \begin{vmatrix} \hat{\lambda}_1 & -2I_N & & \\ -I_N & \hat{\lambda}_1 & -I_N & \\ & \ddots & \ddots & \ddots \\ & -I_N & \hat{\lambda}_1 & -I_N \\ & & -2I_N & \hat{\lambda}_1 \end{vmatrix}$$

The block elements of $\hat{\Lambda}_2$ are diagonal and hence $\hat{\Lambda}_2$ can be reduce to a block diagonal matrix as

(41) $$\hat{\Lambda}_2 = PP\hat{\Lambda}_2PP = P(P\hat{\Lambda}_2P)P = P\hat{T}_2P$$

where $\hat{T}_2 = \text{Diag}\{\hat{T}_{2i}\} \in \mathbb{R}^{N^2 \times N^2}$ and $\hat{T}_{2i} \in \mathbb{R}^{N \times N}$ is a tridiagonal matrix with a structure similar to $\hat{T}$ for which $a = \hat{\lambda}_{1i}$ and $b = -1$. From Theorem 4, the eigenvalue-eigenvector decomposition of $\hat{T}_2$ is given by

(42) $$\hat{T}_2 = \hat{Q}\hat{\lambda}_2\hat{Q}^{-1}$$

The eigenvalue-eigenvector decomposition of matrix $\hat{M}_{L2}$ in Eq. (38) follows by substituting Eqs. (42) and (41) into Eq. (40). 0

Note that the matrix $\hat{Q}$ is not orthogonal. However, as shown in [21], $\hat{Q}$ and $\hat{Q}^{-1}$ can be expressed as

(43) $\hat{Q} = CS$ and $\hat{Q}^{-1} = (2/N-1)S^{-1}C$

where C is the one-dimensional Discrete Cosine Transform (DCT) operator and S is a diagonal scaling matrix as S = [2, 1, ... 1, 2]. Let us define $\hat{\lambda}'_2 = (4\hat{\lambda}_2)/(N-1)^2$, $\mathcal{S} = \text{Diag}[S, S, \ldots S1$, $\mathcal{S}^{-1} = \text{Diag}\{S^{-1}, S^{-1}, \ldots S^{-1}]$, and C = Diag[C, C, ,.., Cl. Equation (38) can be then written as

(44) $$\hat{M}_{L2} = C\mathcal{S}PC\mathcal{S}\hat{\lambda}'_2\mathcal{S}^{-1}CP\mathcal{S}^{-1}C = C\mathcal{S}PC\hat{\lambda}'_2CP\mathcal{S}^{-1}C$$

Similar to the two-dimensional case with Dirichlet boundary condition, the time-parallel algorithm is derived by replacing the expression of matrix $\hat{M}_{L2}$, given by Eq. (44), into Eq. (15).

The matrices $\Psi = C\mathcal{S}PC$ and $\Psi^{-1} = CP\mathcal{S}^{-1}C$ do not represent the operators for the direct and inverse two-dimensional DCT. However, multiplication of any vector by matrix C corresponds to performing N DCTS of size N which, by using the fast techniques, can computed in a time of $0(N^2 \text{Log } N)$. Note that, since the matrices S and S^{-1} are diagonal, the time-parallel algorithm for Neuman boundary condition have a same decoupled structure as that of Section III.

For one-dimensional case the derivation of the time-parallel simply follows from Theorem 4. Also, for three-dimensional case, the derivation of time-parallel algorithm is straightforward and can be carried out in a similar fashion as that in Section IV.

C. Higher-Order Finite-Difference Schemes

The time-parallel algorithms can be also extended to the solution of problem while using higher-order finite-difference schemes. To this end, let us consider the two-dimensional problem of Section III. The five-point finite-difference scheme has a second-order accuracy, i.e., $O(h^2)$. However, if the problem is discretized by using a nine-point finite difference scheme then a fourth-order accuracy, i.e., $O(h^4)$, can be achieved. The discretization of

two-dimensional Laplace operator by using a nine-point finite difference scheme results in a block tridiagonal matrix $\bar{M}_{L2}$ as [231

$\bar{M}_{L2}$ = Tridiag[E, D, E]$\epsilon\mathbb{R}^{N^2xN^2}$ with D = Tridiag[-4, 20, -4]$\epsilon\mathbb{R}^{NxN}$ and

E = Tridiag[-1, -4, -1]$\epsilon\mathbb{R}^{NxN}$. From Theorem 1, it follows that

a) The submatrices D and E have a common set of eigenvectors given by Q, and

b) The eigenvalues of submatrices D and E are given as

λ^D = Diag$\{\lambda_i^D\}\epsilon\mathbb{R}^{NxN}$ with λ_i^D = 20 - 8cos(iπ/N+1), i = 1, 2, ... N

λ_i^E = Diag$\{\lambda_i^E\}\epsilon\mathbb{R}^{NxN}$ with λ_i^E = -4 - 2cos(iπ/N+1), i = 1, 2, ..., N.

Theorem 6. The matrix $\bar{M}_{L2}$ has an eigenvalue-eigenvector decomposition as

$$\bar{M}_{L2} = QPQ\bar{\lambda}_2 QPQ \tag{45}$$

where $\bar{\lambda}_2$ = Diag$\{\bar{\lambda}_{2i}$, $\}\epsilon\mathbb{R}^{N^2xN^2}$, i and j = 1, 2, ... , N, with

$$\bar{\lambda}_{2ij} = 20 - 16\cos(i\pi/N+1) - 4\cos(i\pi/N+1)\cos(j\pi/N+1) \tag{46}$$

Proof. By using the eigenvalue-eigenvector decomposition of submatrices C and D, the matrix $\bar{M}_{L2}$ can be written as

$$\bar{M}_{L2} = \text{Tridiag}[Q\lambda^E Q, Q\lambda^D Q, Q\lambda^E Q] = Q\bar{\Lambda}_2 Q \tag{47}$$

$\bar{\Lambda}_2$ = Tridiag[λ^E, AD, λ^E]$\epsilon\mathbb{R}^{N^2xN^2}$ is a block tridiagonal matrix whose block elements are diagonal and hence it can be reduce to a block diagonal matrix as

$$\bar{\Lambda}_2 = PP\bar{\Lambda}_2 PP = P(P\bar{\Lambda}_2 P)P = P\bar{T}_2 P \tag{48}$$

where $\bar{T}_2$ = Diag$\{\bar{T}_{2i}\}$, i = 1, ... N, and $\bar{T}_{2i}$ = Tridiag [λ_i^E, λ_i^D, λ_i^E]. $\bar{T}_{2i}$ is a symmetric tridiagonal Toeplitz matrix whose eigenvalues are given as

$$\lambda(\bar{T}_{2i}) = \lambda_i^D + 2\lambda_i^E\cos(j\pi/N+1)$$
$$= 20 - 16\cos(i\pi/N+1) - 4\cos(i\pi/N+1)\cos(j\pi/N+1)\ J = 1, 2, ... N$$

The eigenvalue-eigenvector decomposition of $\bar{T}_2$ is given as

$$\bar{T}_2 = Q\bar{\lambda}_2 Q \tag{49}$$

The eigenvector-eigenvalue decomposition of matrix $\bar{M}_{L2}$ in Eq. (45) is then obtained by replacing Eqs. (49) and (48) into Eq. (47). □

The time-parallel algorithm is derived by substituting the eigenvector-eigenvalue decomposition of matrix $\bar{M}_{L2}$ into Eq. (15).

Theorem 6 indicates that the matrices $\tilde{M}_{L2}$ and M_{L2} have a common set of eigenvectors but different sets of eigenvalues. Therefore, the computational structure and cost of the time-parallel algorithm for nine-point scheme are exactly the same as those for five-point scheme. This implies that a higher accuracy in space discretization can be achieved with no additional computation or communication cost.

VI. Some Issues in Practical Implementation of Time-Parallel Algorithms

A. On-line Computation of Diagonal Matrices $(D_j)^i$

In discussing the time-parallel algorithms in previous sections, it was assumed that the diagonal matrices $(D_j)^i$, i = 1, ... M, and j = 1, 2, 3 (J indicates the dimension of problem) can be precomputed. This assumption holds for cases wherein a same problem is solved many times with different initial and/or boundary conditions. For such cases, all $(D_j)^i$ can be precomputed since they are only function of problem size and time and space discretization parameters. Here, we consider the case wherein the problem is solved once and hence the cost of computing $(D_j)^i$ needs to be included in the overall cost.

To begin, note that, the computation of $(D_j)^i$ and the vector $\tilde{U}^{(0)}$ in Step I are completely decoupled and can be performed in parallel. Using $O(MN^J)$ processors, the parallel computation of $(D_j)^i$ from the set of FOHLRS

$$(D_j)^i = D_j(D_j)^{i-1} \qquad i = 1, 2, \ldots, M \tag{50}$$

can be performed in $O(\text{Log } M)$ steps [19] while the computation of the vector $\tilde{U}^{(0)}$ takes $O(\text{Log } N)$ steps. If these two computations are performed in parallel then the overall cost of computing $\tilde{U}^{(0)}$ and $(D_j)^i$ is $\max(O(\text{Log } N), O(\text{Log } M))$. Since with $O(MN^J)$ processors the computations in Steps II and III can be performed in $O(\text{Log } N)$ steps, it then follows that the overall computational complexity of time-parallel algorithms is of $\max(O(\text{Log } N), O(\text{Log } M)) + (\text{Log } N)$, that is, of $O(\text{Log } N) + O(\text{Log } M)$ for M>N, and of $O(\text{Log } N)$ for M<N.

For practical implementation, the parallel computation of $(D_j)^i$ can be

performed with no communication so that the highly decoupled structure of the algorithms is preserved. To this end, consider the parallel implementation of time-parallel algorithms by using M groups of processors. Note that any group of processors, say Group i, needs only to compute $(D_j)^i$ and not all the intermediate powers of D_j. In this case, each group of processors computes the FOHLR in Eq. (50) in a different fashion as follows.

a) For $i = 2^n$, the computation of Eq. (50) is performed as

$$(D_j)^{2^k} = \left((D_j)^{2^{k-1}}\right)^2 \qquad k=1,2,\ldots,\mathrm{Log}_2 i \tag{51}$$

with a serial computational cost of $O(\mathrm{Log}_2 i)$.

b) For $i \neq 2^n$ but $2^n > i > 2^{n-1}$, we can write i as

$$i = \sum_{k=0}^{n-1} a_k 2^k \tag{52}$$

where a_k = 0 or 1. Note that, Eq. (52) describes the binary representation of i which is based on the fact that any integer i, $2^n > i > 2^{n-1}$, can be represented by n bits, For this case, first all $(D_j)^{2^k}$, k = 1, 2, ... (n-l), are computed from Eq. (51) in $O(n-1)$ steps and then $(D_j)^i$ is computed as

$$(D_j)^i = \prod_{k=0}^{(n-1)} (D_j)^{2^k} \tag{53}$$

Note that, in the above product only those $(D_j)^{2^k}$ for which $a_k = 1$ need to be included. For the worst case, i.e., where all $a_k = 1$, Eq. (53) involves the multiplication of n terms and **hence its serial computation can be done in $O(n-1)$ or $O(\lfloor \mathrm{Log}_2 i \rfloor)$ steps. ($\lfloor x \rfloor$ indicates the greatest integer smaller than or equal to x.) The overall computation cost is determined by that of computation of $(D_j)^M$ which for the worst case, i.e., $M = 2^m - 1$,** is of $O(2\lfloor \mathrm{Log}_2 M \rfloor)$ where $\lfloor \mathrm{Log}_2 M \rfloor = m$.

Compared with the direct parallel computation of the FOHLR in Eq. (50), the computation cost is at most increased by a factor of about two. However, this scheme does not require any communication among groups of processors and hence the decoupled structure of the overall computation is preserved.

B. **Performance of M-Parallel Coarse-Grain Implementation**

Our discussions in previous sections have been mainly concerned with the asymptotic performance of time-parallel algorithms. Here, we briefly discuss the coarse-grain implementation of the algorithms on massively parallel MIMD architectures and analyze the expected performance. In this discussion, only two- and three-dimensional problems are considered.

At this point, we need to draw a conclusion regarding the relative serial efficiency of the CR, MD, and FA algorithms. This conclusion forms the basis for the choice of optimal serial algorithm which 1s needed for analyzing the performance of the time-parallel algorithms, For two-dimensional case, the CR algorithm is faster than both the MD and FA algorithms due to a smaller coefficient of $N^2 \text{Log } N$-dependent terms [321. However, one major drawback of the CR algorithm is that its application is restricted by the size of problem, N. Sweet [331 has generalized the CR algorithm for the arbitrary problem size though with a slightly reduced efficiency. For three-dimensional case, as discussed before, the CR algorithm is less efficient than the MD and FA algorithms. It can be also shown that, for both two- and three-dimensional cases, the MD algorithm is faster than the FA algorithm by about a factor of two. Therefore, for both two- and three-dimensional cases, we consider the MD algorithm as the fastest serial algorithm for solution of Eq. (6).

From the above discussion, it follows that a fast alternative for serial solution of the problem is to use the modified C-N method, given by Eqs. (6)-(7), along with the MD algorithm for solution of the linear system in Eq. (6). Using this alternative, it can then be shown that the cost of serial solution of problem, denoted as T_{S1}, is given by

$$T_{S1} = MN^{j-1}(2(j-1)FST + TDS + VA) \tag{54}$$

where FST, TDS, and VA denote, respectively, the cost of one-dimensional fast sine transform of size N, tridiagonal system solution with NxN symmetric Toeplitz coefficient matrix, and addition of two Nx1 vectors. The cost of

serial implementation of time-parallel algorithms, denoted as T_{S2}, is given by

(55) $$T_{S2} = N^{j-1}((M+1)(jFST + DMV) - DMV)$$

where DMV denotes the cost of multiplication of an NxN diagonal matrix by an Nx1 vector, As can be seen, $T_{S2}>T_{S1}$ for j = 2 but $T_{S2}<T_{S1}$ for j = 3. This implies that the time-parallel algorithm is the most efficient even for serial solution of three-dimensional problems.

Now consider the M-parallel implementation of time-parallel algorithms wherein the computation of $\tilde{U}^{(1)}$ and $U^{(1)}$ is assigned to the ith processor. Assuming that Step I is performed in a serial fashion but the computations in Steps II and III are performed in parallel, the computation cost of this M-parallel implementation strategy, T_{MP}, is obtained as

(56) $$T_{UP} = N^{j-1}(2jFST + DMV)$$

The speedup of M-parallel implementation, SP_{MP}, is then given by

$$SP_{MP} = T_{S1}/T_{MP} = M(2FST + TDS + VA)/(4FST + DMV) \qquad j \geq 2$$

$$SP_{MP} = T_{S2}/T_{MP} = ((M+1)(3FST + DMV) - DMV)/(6FST + DMV) \quad j = 3$$

Since TDS, VA, and DMV are of O(N) while FST is of $O(N\text{Log } N)$, SP_{MP} can be approximated as

(57) $$SP_{MP} = M/2 \qquad j \geq 2$$

(58) $$SP_{MP} = (M+1)/2 \qquad j = 3$$

However, the performance of this M-parallel implementation strategy should be also judged by taking into account the communication and synchronization overhead. In fact, this strategy corresponds to a straightforward mapping of the Figs. lb and 1c and, as can be seen, the only communication activity involved is the broadcasting of vector $\tilde{U}^{(0)}$ to all processors. It also represents a highly coarse-grain parallel computation strategy in which each processor performs a series of computations in Steps 11 and 111 with a total cost, of $N^{j-1}(jFST + DMV)$ asynchronously and without any need to communicate with other processors. The simple communication structure and highly coarse-grain size make the M-parallel implementation strategy very efficient for

massively parallel MIMD architectures. In particular, it is highly suitable for a new class of emerging MIMD architectures, represented by the Intel's Touchstone Delta and Paragon, which provide a large number of powerful processors but a rather simple and limited communication structure) i.e., a mesh communication structure [34].

The above strategy, while being the simplest and most straightforward, is not the most efficient since it is based on the assumption that, though M processors are available, the computation of Step I is done in a serial fashion. However, the computation of Step I corresponds to performing a two- or three-dimensional DST and as such it offers a high degree of parallelism which can be exploited to increase the overall speedup (see [35] or many references given in [211). This can be better seen by reexamining the computations involved in Step I in more detail. For J = 2, the computation in Step I corresponds to performing a two-dimensional DST as

a. $\tilde{U} = Q U^{(0)}$

b. $\underline{U}^{(0)} = P\tilde{U}$

c. $\tilde{U}^{(0)} = Q\underline{U}^{(0)}$

The N one-dimensional FSTS in Substeps a and c are completely decoupled and can be performed in parallel while Substep c involves a global communication. For j = 3, the computation in Step I corresponds to performing a three-dimensional DST as

a. $\tilde{U} = W^{(0)}$

b. $\underline{U}^{(0)} = P\tilde{U}$

c. $\tilde{U}^{(0)} = Q\underline{U}^{(0)}$

In Substep a, N two-dimensional FSTS can be computed in parallel. There is an even higher degree of parallelism in computation of Substep c since N^2 one-dimensional FSTS can be performed in parallel. However, Substep b again involves a global communication.

Let us assume that a speedup of K is achieved in the computation of Step

1, T_{MP} and SP_{MP} are then given by

(59) $T_{MP} = N^{j-1}(jFST/K + jFST + DMV)$

(60) $SP_{MP} = M/(1 + l/K) = MK/K+1 \qquad j \equiv 2$

(61) $SP_{MP} = (M+1)/(1 + I/K) = (M+1)K/K+1 \qquad j = 3$

Interestingly, Eqs. (60)-(61) suggest that even a limited speedup in computing Step I will result in a rather significant increase in the overall speedup, SP_{MP}, For example, a speedup of K = 3 will result in a 25% increase in SP_{MP}. For large K, say K>10, SP_{MP} will be very close to M, which indicates a linear speedup in the computation. For large problems, i.e. , large N or J = 3, such a speedup in the computation of Step I seems to be easily achievable even on parallel architectures with simple communication structures.

The above analysis indicates that, under realistic assumptions, the M-parallel implementation of time-parallel algorithms on the MIMD architectures can result in a speedup close to the linear one. The minimum communication and synchronization requirements of the time-parallel algorithms also suggests that with any number of processors smaller than M it is more efficient to exploit parallelism in time rather than in space.

C. Time- and Space-Parallel Implementation

Many emerging MIMD parallel architectures, e.g., Intel's iPSC/860, Touchstone Delta, and Paragon, use powerful vector processors such as Intel i860 as the node processor. Another advantage of the time-parallel algorithms for implementation on these architectures is that the computation performed by each processor can be efficiently vectorized to exploit the node vector processing capability and hence increase the overall computational speed. To see this, note that, the computation in Step 11 is already in a form highly suitable for vector computation. Many algorithms have been developed for efficient vector computation of fast transforms (see for example [36] or the references in [21]). An even greater efficiency in vector computation of Steps

I and III can be achieved by noting that each processor has to perform a series of decoupled FSTs. Therefore, the computation can be organized in a way to further increase the efficiency for vectorization [35,36].

With a number of processors greater than M further speedup can be achieved by exploiting space-parallelism in computing Steps II-III. Again, note that, parallel computation of Step II is straightforward. Exploitation of space-parallelism in computing Step III corresponds to parallel computation of two- or three-dimensional DST which was discussed above.

VII. Discussion and Conclusion

In this paper, we developed time-parallel algorithms for solution of a class of parabolic PDEs defined by Eq. (l). The basic idea in our approach is to use a transformation based on the eigenvalue-eigenvector decomposition to diagonalize the matrices involved in the time-stepping iterations given by Eqs. (3)-(5). This diagonalization results in a decoupling of the iterations which in turn allows the solution for all the time steps to be computed in parallel. The time-parallel algorithms achieve maximum parallelism in time since their complexity is either of O(Log M) or is independent of M

The time-parallel algorithms for one-, two-, and three-dimensional cases have a similar structure for parallel computation. However, they differ in their efficiency for serial computation. The time-parallel algorithm for one-dimensional case is highly inefficient for serial computation, while the one for three-dimensional case seems to be the most efficient for serial computation. This implies an optimal efficiency for parallel computation for three-dimensional case since the algorithm not only provides a high degree of parallelism but it does so by also reducing the total number of operations.

We also developed time-parallel algorithms for solution of problem with Neumann boundary condition as well as by using higher-order finite-difference schemes, For the latter case, it was shown that a higher accuracy in space

discretization can be achieved with no additional computation or communication cost. If a large number of processors are available then a higher accuracy in time discretization can be also achieved by reducing the time step size At (and hence increasing M) with a small increase in the computational cost. Note that, with a sufficient number of processors,' the computation of step III is independent of M while that of Step II, for the worst case, may be increased by $O(\text{Log } M)$.

Our results clearly show that, unlike the general assumption, the time-stepping iterations for the class of problems defined by Eq. (1) can be fully parallelized in time. However, the time-parallel computing approach can be also applied to a wider class of problems, The extension of time-parallel algorithms to the solution of linear inhomogeneous parabolic PDEs with constant and variable coefficients is presented in [37]. The generalization of time-parallel computation approach for solution of a more general class of evolutionary PDEs, including both parabolic and hyperbolic PDEs, on irregular domains is reported in [381.

ACKNOWLEDGMENT

The research described in this paper was *performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration (NASA). I am highly indebted to my colleagues Drs. J. Barhen and N. Toomarian for many insightful discussions, suggestions, and encouragement.*

REFERENCES

[11 R. S. Varga, *Matrix Iterative Analysis*, Prentice-Hall, NJ, 1962.

[2] P. N. Swarztrauber and R. A, Sweet, "Efficient Subroutines for the Solution of General Elliptic and Parabolic Partial Differential Equations, " Atmospheric Technology, pp. 79-81, Sept. 1973,

[3] J.M. Ortega and R.G. Voigt, *Solution of Partial Differential Equations on Vector and Parallel Computers,* SIAM Pub., 1984.

[4] E. Gallopoulos and Y. Saad, "On the Parallel Solution of Parabolic Equations, " Proc. ACM Int. Conf. on Supercomputing, pp. 17-28, June 1989,

[5] S. M. Serbin, "A Scheme for Parallelizing Certain Algorithms for the Linear Inhomogeneous Heat equation, " SIAM J. Sci. Stat. Comput. ,

Vol. 13(2), pp. 449-458, March 1992.

[6] G. Rodrigue, "A Parallel First-Order Method for Parabolic Partial Differential equations, " in *High-Speed Computation,* J. S. Kowalik (Ed.), pp. 329-342, Springer-verlag, 1984.

[71 G. Rodrigue and D. Wolitzer, "Preconditioned Time-Differencing for the Parallel Solution of the Heat Equation, " Proc. 4th SIAM Conf. on Parallel Processing, pp. 268-272, 1990.

[81 D. J. Evans, ''Alternating Group Explicit Methods for the Diffusion Equation, " Appl. Math. Modelling, Vol. 9, pp. 201-206, 1985.

[91 E. Lelarasmee, A. Ruheli, and A. L. Sangiovanni-Vincentelli, ''The Waveform Relaxation Method for the Time Domain Analysis of Large Scale Integrated Circuits, " IEEE Trans. Computer-Aided Design, Vol. 1, pp. 131-145, 1982,

[101 J. H. Saltz and V. K. Nail, ''Towards Developing Robust Algorithms for Solving Partial Differential Equations on MIMD Machines, " Parallel Computing, Vol. 6, pp. 19-44, 1988.

[111 D. E. Womble, "A Time-Stepping Algorithm for Parallel Computers, " SIAM J. Sci. Stat. Comput. , Vol. 11(5), pp. 824-837, 1990.

[121 W. Hackbusch, ''Parabolic Multigrid Methods, " Proc. 6th Int. Symp. on Computing Methods in Applied Sciences and Engineering, Dec. 1983.

[131 G. Horton and R. Knirsch, "A Time-Parallel Multigrid-Extrapolation Method for Parabolic Partial Differential Equations, " Parallel Computing, vol. 18, pp. 21-29, 1992.

[141 G. Strang and G. J. Fix, *An Analysis of the Finite Element Method,* Prentice-Hall, Englewood Cliffs, NJ, 1973.

[151 A. Fijany, "Time Parallel Algorithms for Solution of Linear Parabolic PDEs , " Jet Propulsion Lab. Eng. Memorandum, EM 347-93-002, Feb. 1993.

[161 G. Smith, *Numerical Solution of Partial Differential Equations.* Clarendon Press, Oxford, 1985.

[171 R.F. Boisvert, ''Algorithms for Special Tridiagonal Systems, " SIAM J. Sci. Stat. Comput. , Vol. 12(2), pp. 423-442, March 1991.

[181 A. H. Sameh, S. C. Chen, and D. J. Kuck, "Parallel Poisson and Biharmonic Solvers, " Computing, Vol. 17, pp. 219-230, 1976.

[191 R. Hockney and C. Jesshope, *Parallel Computers.* Adam Hilger Ltd., 1981.

[20] S. Barnett, *Matrices; Methods and Applications.* Clarendon Press, 1990,

[211 C. Van Loan, *Computational frameworks for the Fast Fourier Transform,* SIAM, Philadelphia 1992.

[221 0. Buneman, "A Compact Non-Iterative Poisson Solver, " Rep. 249, Stanford University Institute for Plasma Research, Stanford, California, 1969.

[23] B. Buzbee, G. Golub, and C. Nielson, "On Direct Methods for Solving Poisson Equations, " SIAM J. Numer. Anal., Vol. 7, pp. 627-656, 1970,

[241 B. Buzbee, "A Fast Poisson Solver Amenable to Parallel Computation, " IEEE Trans. Computers, Vol. C-22, pp. 793-796, 1973.

[25] R. Hockney, "A Fast Direct Solution of Poisson's Equation Using Fourier Analysis, " J. ACM, Vol. 12, pp. 95-113, 1965.

[26] R. Sweet, "A Parallel and Vector Variant of the Cyclic Reduction Algorithm, " SIAM J. Stat, Sci. Comput. , Vol. 9, pp. 761-765, 1988.

[271 E. Gallopoulos and Y. Saad, "A Parallel Block Cyclic Reduction Algorithm for the Fast Solution of Elliptic Equations, " Parallel Computing, vol. 10, pp. 143-159, 1989.

[281 P. Swarztrauber and R. Sweet, "Vector and Parallel methods for the Direct Solution of Poisson's Equation, " J. Computional & Applied Math., Vol. 27, pp. 241-263, 1989.

[29] A. Sameh, "A fast Poisson solver for multiprocessors, " *Elliptic Problem Solvers II*, G. Birkhoff and A. Schoenstadt (Eds.), Academic Press, 1984.

[30] R. Wilhelmson and J. Ericksen, "Direct Solution for Poisson's Equation in Three Dimensions" J. Comp. Physics, Vol. 25, pp. 319-331, 1977.

[31] R. Sweet, W. Briggs, S. Olivera, J. Porsche, and T. Turnbull, "FFTs and Three-Dimensional Poisson Solvers for Hypercube, " Parallel Computing, vol. 17, pp. 121-131, 1991.

[321 C. Temperton, "Direct Methods for the Solution of the Discrete Posisson Equation: Some Comparisons, " J. Comp. Physics, Vol. 31, pp. 1-20, 1979.

[331 R. Sweet, "A Cyclic Reduction Algorithm for Solving Block Tridiagonal Systems of Arbitrary Dimension, " SIAM J. Numer. Anal., Vol. 14(4), pp. 706-720, 1977.

[34] R. Hockney and E. Carmona, "Comparison of Communications on the Intel iPSC/860 and Touchstone Delta," Parallel Computing, Vol. 18, pp. 1067-1072, 1992.

[35] P. Swarztrauber, ''Multiprocessor FFTs," Parallel Computing, Vol. 5, pp. 197-210, 1987.

[36] D. Bailey, "A High Performance Fast Fourier Transform Algorithm for the CRAY-2, " J. of Supercomputing, Vol. 1, pp. 43-60, 1987.

[37] A. Fijany, ''Time-Parallel Algorithms for Solution of Linear Inhomogeneous Parabolic PDEs with Constant and Variable Coefficients, " Submitted to SIAM J. Sci. Stat. Comput. .

[381 A. Fijany, "On the Structure of Time-Parallel Algorithms for Solution of Linear Evolutionary Partial Differential Equations, " In preparation.

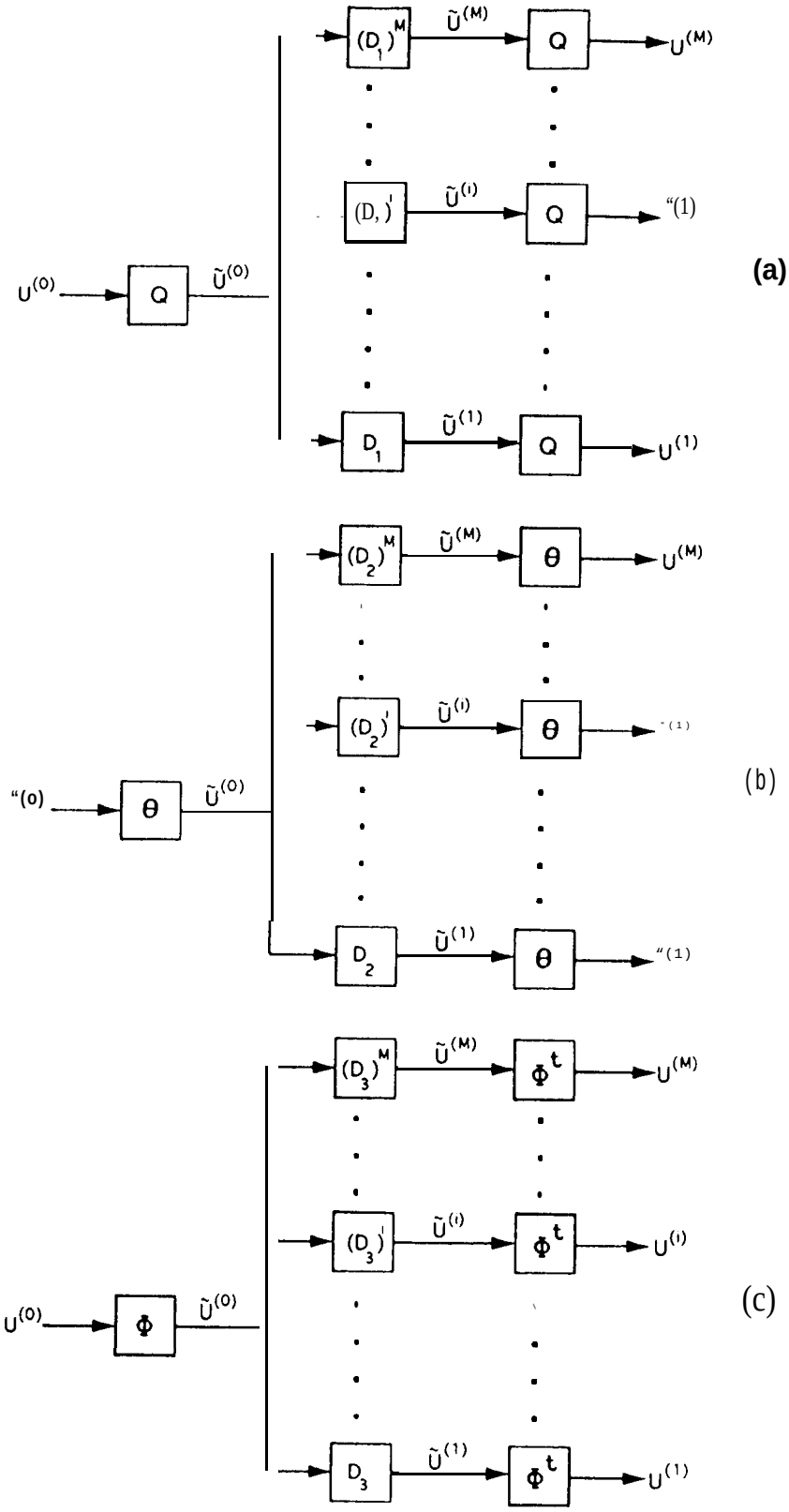

Figure 1. Computational Structure of Time-Parallel Algorithms. a: One-Dimensional Problem. b: Two-Dimensional problem. c: Three-Dimensional Problem.